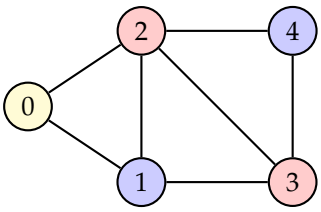
Problème 1. Coloration de graphes

1.1 Calcul du nombre chromatique

Dans ce problème, on considère des graphes non-orientés et **sans boucles**. Le problème que l'on cherche à résoudre est le problème de la coloration de graphe. On veut attribuer à chaque sommet une couleur de manière à ce que deux sommets adjacents (*i.e.* reliés par une arête) aient des couleurs différentes. Le but étant de colorier le graphe en utilisant le moins de couleurs différentes possibles. La définition ci-dessous formalise cette notion.

Définition Soit $G = (S, A)$ un graphe à $n \geq 1$ sommets. Pour $k \geq 1$, une *k-coloration* c de G est une application de S dans $\{0, \ldots, k-1\}$. Une k-coloration c est *valide* si deux sommets adjacents ont deux couleurs distinctes : pour toute arête $\{u, v\} \in A$, $c(u) \neq c(v)$.



$c(0) = 0$
$c(1) = 1$
$c(2) = 2$
$c(3) = 2$
$c(4) = 1$

FIGURE 1 – Exemple de 3-coloration non valide, car les sommets 2 et 3 ont la même couleur.

Question 1.1. Exemple

(a) Donner une 3-coloration valide du graphe de l'exemple ci-dessus.

(b) Montrer qu'il n'existe pas de 2-coloration valide de ce graphe.

Question 1.2. Nombre chromatique Le plus petit $k \geq 1$ tel le graphe G admette une k-coloration valide est appelé le *nombre chromatique* de G et est noté $\chi(G)$. Dans la suite de ce problème, on s'intéresse à calculer $\chi(G)$, éventuellement de manière approchée.

(a) Quel est le nombre chromatique du graphe de la figure 1 ?

(b) Montrer que pour tout graphe G à n sommets, $\chi(G) \leq n$.

Pour tout $n \geq 3$, on note C_n le cycle de longueur n : l'ensemble de ses sommets est $\{0, \ldots, n-1\}$ et l'ensemble de ses arêtes est $\{\{0,1\}, \{1,2\}, \ldots, \{n-2, n-1\}, \{n-1, 0\}\}$.

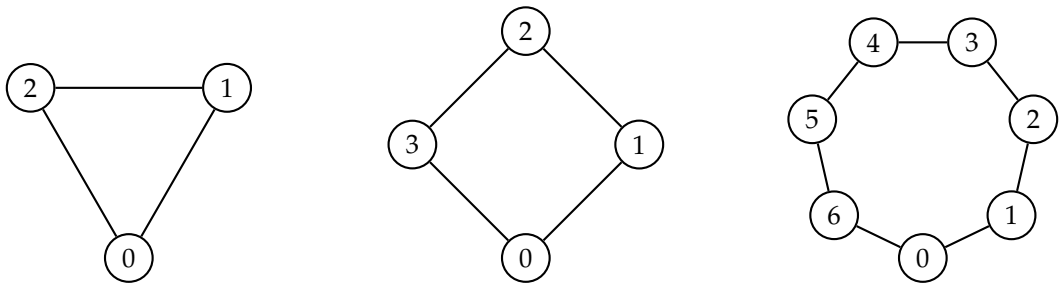


FIGURE 2 – Les cycles de longueur 3, 4 et 7.

(c) Donner la valeur du nombre chromatique de C_n pour tout $n \geq 3$.

Question 1.3. Recherche exhaustive On s'intéresse au problème de tester si un graphe admet une k-coloration. Un algorithme naïf pour résoudre le problème consiste à parcourir toutes les k-colorations possibles et à tester la validité de chacune. Le but de cette question est d'écrire cet algorithme en Python.

La représentation représentation des graphes non-orientés est donnée par la classe `Graphe` de la figure 3. Les sommets du graphe sont les entiers de 0 à $n-1$, et la liste `_voisins` contient, pour chaque sommet, la liste de ses voisins.

On choisit de représenter une k-coloration d'un graphe de sommets $\{0, \dots, n-1\}$ par une liste à n éléments dont la i-ème valeur est la couleur du sommet i.

```
class Graphe:
    def __init__(self,n):
        "Constructeur, n est le nombre (fixe) de sommets"
        self._n = n
        self._voisins = [[] for _ in range(n)]

    def sommets(self):
        "Renvoie la liste des sommets"
        return list(range(self._n))

    def voisins(self, i):
        "Renvoie la liste des voisins du sommet i"
        return self._voisins[i]

    def ajoute_arete(self,i,j):
        "Ajoute l'arête {i,j}, en supposant qu'elle n'existe pas."
        self._voisins[i].append(j)
        self._voisins[j].append(i)
```

FIGURE 3 – Représentation des graphes par listes d'adjacence.

Les méthodes demandées ci-dessous sont à écrire comme méthodes de la classe `Graphe`.

(a) Écrire une méthode `est_valide(self, c)` qui prend une coloration `c` et renvoie un booléen indiquant si la coloration `c` est une coloration valide du graphe.

(b) Écrire une méthode `liste_colorations(self, k)` qui prend en entrée $k \geq 1$ et renvoie une liste contenant toutes les k-colorations possibles du graphe.

(c) Écrire une méthode `est_coloriable(self, k)` qui prend en argument un entier $k \geq 1$ et renvoie un booléen indiquant si le graphe est k-coloriable.

(d) Donner la complexité de l'algorithme en fonction de n, le nombre de sommet du graphe, et de k.

(e) Écrire une méthode `chromatique(self)` qui calcule le nombre chromatique du graphe.

(f) On considère le problème COLOR suivant : étant donné un graphe G et un entier k, G est-il k-coloriable ? Montrer que COLOR appartient à la classe de complexité NP.

Question 1.4. Graphes 2-coloriables

(a) Montrer qu'un graphe admet une 2-coloration si et seulement s'il est biparti, c'est-à-dire s'il existe une partition $S = X \sqcup Y$ de ses sommets en deux sous-ensembles telle que toute arête de G a une extrémité dans X et l'autre dans Y.

(b) Écrire une méthode `deux_coloration(self)` qui calcule une 2-coloration du graphe s'il en existe une, et soulève une exception de type `ValueError` sinon.

1.2 Estimation du nombre chromatique

Déterminer le nombre chromatique d'un graphe étant un problème difficile, on utilise une heuristique pour approcher ce nombre. Pour cela, on fixe un ordre sur les sommets et on applique l'algorithme

glouton suivant : on parcourt les sommets dans l'ordre fixé, et on colorie chaque sommet avec la plus petite couleur qui ne crée pas de conflit avec les voisins déjà coloriés.

Pour implanter cet algorithme, on manipule des colorations *partielles* du graphes (certains sommets n'ont pas encore de couleur). Pour cela, on utilise toujours une liste dont la $i^{\text{ème}}$ case représente la couleur du sommet i, avec la valeur `None` quand la couleur n'est pas encore définie.

Question 1.5. Coloration gloutonne

(a) Colorier le graphe de la figure 1 en utilisant cette heuristique avec l'ordre $0 < 1 < 2 < 3 < 4$, puis avec l'ordre $0 < 4 < 3 < 1 < 2$.

(b) Écrire une méthode `couleur_libre(self, t, u)` qui prend une coloration partielle t et un numéro de sommet u non encore colorié et qui renvoie la plus petite couleur que l'on peut lui affecter sans créer de conflit avec ses voisins déjà coloriés.

On représente un ordre sur $\{0, \dots, n-1\}$ par une liste `o` de taille n dont la $i^{\text{ème}}$ case contient le $i^{\text{ème}}$ élément de l'ordre. Par exemple, le deuxième ordre de la question (a) est représenté par la liste `[0,4,3,1,2]`.

(c) Écrire une méthode `coloration(self,o)` qui prend en argument un ordre `o` sur les sommets du graphe et renvoie la coloration obtenue en utilisant l'heuristique décrite précédemment.

Le degré d'un sommet u d'un graphe $G = (S, A)$, noté $d(u)$, est le nombre de ses voisins. On note $\Delta(G) = \max_{u \in S} d(u)$ le degré maximal des sommets de G.

(d) Montrer que quelque soit l'ordre, le nombre de couleurs utilisées par l'algorithme glouton est inférieur ou égal à $\Delta(G) + 1$.

(e) En déduire que $\chi(G) \leq \Delta(G) + 1$.

Question 1.6. Heuristique du degré On choisit maintenant un ordre particulier pour l'algorithme glouton : on ordonne les sommets par degrés décroissants.

(a) Écrire une méthode `sommets_tries(self)` qui renvoie la liste des sommets classés par ordre décroissant de degré. *Cette méthode ne doit pas utiliser la fonction `sorted` de Python (ni la méthode `sort` des listes). On pourra construire une liste dont la $i^{ème}$ case contient la liste des sommets de degré i.*

Donner la complexité de la méthode en fonction du nombre n de sommets du graphe.

(b) Écrire une méthode `coloration_par_degres(self)` qui implante l'heuristique décrite ci-dessus.

On justifie maintenant le choix des degrés décroissants dans l'algorithme glouton. On considère dans un premier temps un ordre quelconque. On note $s_1, \dots, s_n$ les sommets de G dans l'ordre fixé, c'est-à-dire que $s_i < s_{i+1}$ pour tout i. On commence par améliorer le résultat de la question 1.5.

(c) Montrer qu'à l'étape i de l'algorithme, le sommet s_i à colorier possède au plus $\min(d(s_i), i-1)$ voisins déjà coloriés.

(d) En déduire que le nombre de couleurs utilisées par l'algorithme glouton est borné par

$$\max_{1 \leq i \leq n} \min\left(d(s_i) + 1, i\right).$$

(e) Montrer que cette borne est minimale si les sommets $s_1, \dots, s_n$ sont triés par ordre de degrés décroissants. *Indication : on peut étudier ce qui se passe quand on échange deux sommets $s_i < s_j$ dans l'ordre, si $d(s_i) < d(s_j)$.*

1.3 Utilisation de la coloration

La coloration de graphes est un problème très important car il permet de modéliser beaucoup de situations. Cette partie présente quelques exemples.

Question 1.7. Modélisation Pour les différentes situations suivantes, définir un graphe G de telle sorte que le problème à résoudre soit équivalent à un problème de coloration du graphe G défini. *Dans chaque cas, le graphe G doit être décrit par son ensemble de sommets et son ensemble d'arête, et l'équivalence entre le problème étudié et la coloration du graphe doit être démontrée.*

(a) Dans le laboratoire de chimie sont entreposés des stocks de produits chimiques dans des armoires. Certains produits ne doivent surtout pas entrer en contact car cela provoquerait une explosion. Par mesure de sécurité, on décide que deux produits qui ne doivent surtout pas entrer en contact doivent être rangés dans deux armoires séparées. L'objectif est de trouver le nombre minimal d'armoires nécessaires pour entreposer de manière sûre l'ensemble des produits dont on dispose.

(b) On souhaite organiser les examens de la Faculté des Sciences. Deux UE ne peuvent pas être planifiées en même temps si elles ont des étudiants en commun [1]. L'objectif est d'utiliser le moins de créneaux horaires différents possibles pour placer tous les examens. L'objectif est de trouver le nombre minimal de créneaux nécessaires pour organiser les examens.

(c) Avec la même situation que précédemment, on cherche à planifier les examens sur deux semaines, avec un créneau par demi-journée. À quelle question de coloration correspond cet objectif ?

1. Autrement dit, on ne demande pas aux étudiants de passer deux examens en même temps !